Finding Related Entities by Retrieving Relations: UIUC at TREC 2009 Entity Track

V.G.Vinod Vydiswaran, Kavita Ganesan, Yuanhua Lv, Jing He, ChengXiang Zhai
Department of Computer Science
University of Illinois, Urbana, IL, USA
{vgvinodv, kganes2, ylv2}@illinois.edu, peaceful.he@gmail.com, czhai@cs.uiuc.edu

ABSTRACT

Our goal in participating in the TREC 2009 Entity Track was to study whether relation extraction techniques can help in improving accuracy of the entity finding task. Finding related entities is informational in nature and we wanted to explore if inducing structure on the queries helps satisfy this information need. The research outlook we took was to study techniques that retrieve relations between two entities from a large corpus, and from those, find the most relevant entities that participate in the given relation with another given entity. Instead of aiming at retrieving pages about specific entities, we tried to address the problem of directly finding the entities from the text. Our experimental results show that we were able to find many related entities using relation-based extraction, and ranking entities based on further evidence from the text helps to a certain extent.

1. PROBLEM FORMULATION

The TREC 2009 Entity Track concentrated on finding related entities. Given an entity of focus, the nature of relation between this entity and other entities, and some information of the type of other entities, the goal was to find all the related entities.

Instead of taking a search perspective looking for homepages of related entities, we wanted to explore the information seeking aspects of the query. Our general goal for the Entity track was to study the usefulness of information extraction techniques in improving the accuracy of the entity finding task. In particular, we focused on investigating whether we can improve accuracy by formulating such entity-finding queries as a relation query, which can be answered through relation extraction. We formulated the query, described by the narrative, as `[entity - relation - entity]` triplet, where relation and the first entity are given, and the type of the other entity is known. The task is to then find instances of the other entity that satisfy the relation. Such a formulation also reflects many other semantic search applications such as redacting (anonymizing) sensitive information, question answering, and intelligence gathering applications.

This formulation was similar to a relation retrieval problem, explored earlier in [7]. A relation is assumed to be binary verb predicate over entities, where the entities can potentially have roles in which they participate in the relation. For example, the relation of touring a new city can be modeled as a person visiting a location. Here, the person always fills in the first slot and the location fills in the second. Special filters can additionally be applied on the two slots to restrict specific subsets of persons or locations (such as a city or a country). In this year's task at TREC, one entity (usually the "first slot") is deterministic and fixed and the second entity is restricted by the type of the named entity. The relation was specified in the narrative, or had to be derived from there.

In the next sections, we describe our approach in detail. We start with an overall system architecture in section 2, detailing the core modules in sections 2.1, 2.2, and 2.3. We explain the parameters of our submitted runs in section 3 and briefly summarize our preliminary evaluation in section 4. Finally, we discuss some challenges we faced in section 5.

2. SYSTEM ARCHITECTURE

Our basic approach can be summarized by the following stages:

1. Formulate a structured query based on the given entity and the relation expressed by the narrative.
2. Retrieve relevant snippets that match the relations specified in the query.
3. Identify named entities in the resultant snippets using state-of-the-art NE taggers (for persons and organizations) and product identifiers.
4. Rank retrieved entities and find homepages for the entities.

We further extend this basic approach in two ways:

- Before the retrieval step, we expand the relation expressed in the given query with semantically similar and related words, derived from WordNet and other linguistic resources such as distributional similarity; resulting in an expanded query for retrieval.
- After retrieval, we explore techniques to improve the accuracy of extraction by searching for more relations that link the two entities in similar contexts, from the corpus. This is expected to help us improve our entity identification task and improve the relative ranking of relevant entities.

The following sections explain the three steps in our model, viz. query formulation and retrieval, entity recognition, and entity filtering and re-ranking.

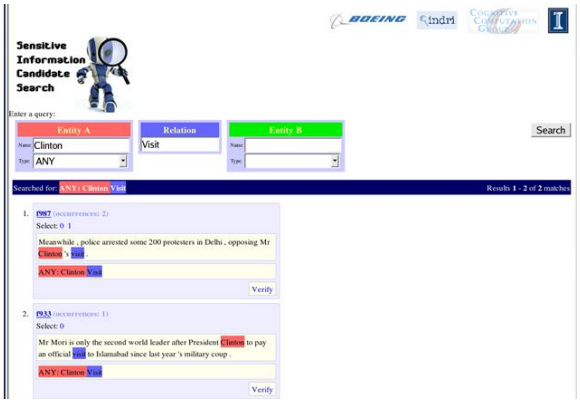

Figure 1: Screen shot of a sample search system [7] that searches for related entities given the relation and one of the entities.

2.1 Relation Retrieval

As we described earlier, we model the information need as a structured query. We identify three parts to the structure: the relation, the entity of focus, and the entity of interest. A sample snapshot of the query interface from a demo system is shown in Fig. 1. This formulation does not describe an arbitrary relation, but we postulate that any long-distance relation could be modeled as a series of multiple binary relations. For example, the task of finding "all team-mates of `Michael Schumacher`" could be addressed by formulating two related relation queries: "`[Michael Schumacher] [drives for] [ORG-1]`" and "`[PER] [drives for] [ORG-1]`", to find related entities through `ORG-1` (the team), which is as yet unknown, but would hopefully get filled in by "`Ferrari`" during execution. Such a formulation would still not capture all pieces of information or relations, such as co-occurrence relation or "is-a" (sub-class) relation (e.g., sportspersons, German nationals, etc.). We plan to treat these as filters that would need external "world knowledge" to solve.

One of the primary challenges is to recognize the relation. For all TREC queries, we use the narrative to derive what the intent of the query is, and hence decide on the relation. In most cases, we could parse the narrative to identify the verb directly or indirectly, when the verb is present in nominalized form (i.e. in noun form). In cases where parsing failed to recognize the relation, simple rewrite rules such as converting the "headline-style" narrative into a grammatical sentence helped. For example, the narrative for query number 1 in test topics, "`Carriers that Blackberry makes phones for.`" was preceded by "`There are many`" to construct a valid grammatical sentence and help parse the narrative better. We restricted our model to a single relational query in the current runs.

Once the relation predicate was identified, we augment it with its synonyms from WordNet [4]. We also incorporated similar words found using distributional similarity computed over a large text corpus. Other works in literature [3, 5] have shown that words belonging to word-classes built in such an unsupervised fashion have high similarity and relatedness among them.

Finally, the given entity is used to restrict the searches to pertain to a primary entity. This way, we hope to retrieve all relations expressed in different surface forms, but related to the specific entity; thereby improving recall without losing too much on precision. We also add other noun phrases from the narrative to the final query as optional keywords. We built a retrieval system over the ClueWeb09 corpus using the Indri toolkit [8], and utilized the rich IndriQuery querying language to query the index. The final query formulation had the primary entity, the relation word with its synonyms, optional noun phrases from the narrative, and the type information of the desired entity, combined in such a way that the retrieval system enforces matching of the primary entity and one of the relation words in all retrieved documents.

2.2 Identifying Entities

The next step is to find the entities participating in the relation of interest. From the relevant documents returned by the retrieval system, we first identify relevant snippets. A snippet is primarily a passage of 2–3 sentences around the words that match the query. Since our queries have both entity and relation components, we hypothesize that the snippets would also contain the second entity that is related to the given entity. The retrieved snippets are passed through state-of-the-art NER taggers and product extractors to identify the entities in the snippets.

2.2.1 Names and Organizations

Researchers have been dealing with named entities for over a decade now, and have proposed and evaluated many models to this effect [2, 6]. Like most NLP tasks, these models derive local and global features based on the actual named entity itself and the context in which it appears; and train a classifier to recognize and classify an entity from free text, using external resources when available. Recognizing named entities is now considered one of the pre-processing steps to analyzing text for deeper semantics, and many NLP toolkits are available for entity recognition.

We used the LBJ-based Named Entity Tagger[1] to tag the retrieved snippets with named entities, viz. person names (`PER`), organizations (`ORG`), locations (`LOC`), and other classes (`MISC`). This state-of-the-art tagger [6] uses non-local features, external knowledge such as gazetteers derived from Wikipedia, and features based on word cluster hierarchy derived using distributional similarity over a larger text corpus. The tagger has also been shown to perform well on Webpages, where most named entities have less context around them.

2.2.2 Products

For the TREC task, we were primarily concerned with person names, organizations, and product names. Since the state-of-the-art named entity recognizer that we used only identified entities such as person, locations, and organizations, we implemented a fairly simple tool to recognize products. Our product recognizer uses a rule based approach and is context independent. It mainly relies on a dictionary of company names and a pre-defined set of patterns for product recognition.

[1]From `http://L2R.cs.uiuc.edu/~cogcomp/software.php`

The search for product names starts with the generation of a set of candidate phrases. Candidate phrases are phrases that match a pre-defined set of regular expression patterns. These candidate phrases could eventually turn out to be true product names. Applying a regular expression pattern, such as "find capitalized phrases containing some numbers with length greater than two", on the text "`The Nokia 6600 was one of the oldest models.`" will result in "`The Nokia 6600`" being tagged as a potential product name.

To be certain that candidate product names found through regular expression matches are indeed true product names, further analysis is needed. For the example provided, the candidate phrase, "`The Nokia 6600`" will be further analyzed with validation and refinement steps, as follows:

- Removal of stop words at the beginning of a candidate product name
- Phrase length validation (varies based on matched pattern)
- Occurrence of company name within a candidate phrase

Various regular expression patterns were defined to capture different types of products with the most specific pattern being at the top. If two patterns match overlapping candidate phrases, the longer phrase is always chosen although this phrase is subject to further validation. Although regular expression matches alone may be sufficient in identifying a product name, to avoid false positives or wrong start and end boundaries it is necessary to validate them through checks like the ones mentioned above. In our implementation, the types of refinement and validation checks performed depend on the regular expression patterns matched.

The product class, in itself, is a heterogeneous mix of multiple classes, depending on the categories they belong to. A music album could have any generic name, whereas a laptop model has a more generalizable name. A pharmaceutical product name, such as "`Synagis®`", is not a company name but is simply a capitalized word ending with a symbol. Such a name will not be correctly identified as a true product name if the above refinement steps are used. Thus, different types of products will need different forms of validation. This is one of the major problems that we faced in recognizing products. Products of different genres have very different ways of being defined – common consumer products often contain model numbers or company names within them, pharmaceutical products tend to be single word names, names of music albums are simply plain text often capitalized, and so on. Thus, we feel that a better approach would be to first identify the origin domain of the text to be tagged (e.g., pharmaceutical, music, journal, etc.), and then apply tagging rules that are specific to that domain. Extending state-of-the-art NE taggers to new classes is indeed a potential research direction to be explored, as has been done in bio-medical and chemical domains. However, for this task, we decided to go with the simpler approach of applying a general set of rules that would capture most common product names with refinement steps specific to the matched regular expression pattern.

2.3 Filtering and re-ranking entities

Once the entities are identified, we try to find which entities are relevant to the query. As of now, the only filter we have applied is that the candidate entities identified are co-located with the entity of interest. We apply the following filter rules to shorten the list of candidate entities.

1. We first drop the entity that has high lexical overlap with the given entity in the query.
2. We look at the type of the desired entity from the query topic and remove all entities that are not of that type. The queries looking for persons and organizations directly correspond to `PER` and `ORG` classes of the NE tagger. But for products, we consider input from Product identifier and the "other" `MISC` class tagged by the LBJ-based NE tagger. This way, we hope to cover a broader class of product names.
3. We also try to prune candidates based on semantic structure of the sentences. The idea is to include candidates from only those sections of the sentences that are part of the SRL predicate-argument structure corresponding to the relation word. However, we found that this rule was very restrictive, in that it implicitly assumed that all entity-relation matches occurred in well-formed sentences (which was not the case for many queries). So, the rule ended up removing many valid occurrences.
4. Finally, we compute string edit-distance between remaining entities; and merge candidates that have high similarity. This heuristic also tries to merge candidate people names that specify only the last name. For example, this assimilation was important in case of `Karen Spärck-Jones`, an "`ACM Athena Award` winner" (query number 2 in test topics). This way, we could handle slight variations in her name, like `Karen Spärck-Jones`, `Karen Sparck Jones`, and `Sparck-Jones`, and treat them all as one.

The next step was to score the candidate entities and get a ranked list of candidates. We could try out two measures for scoring and re-ranking in our runs. These are explained below.

1. **Support** in the retrieved snippets. Support was computed by simply counting the number of times a candidate name appeared in the retrieved snippets. If two candidate names were merged because of small spelling variations, their counts were added up.
2. **Relatedness** with the given entity. We issue multiple queries, one per candidate, that look for possibly other relations between the two entities (i.e. the given entity and one candidate entity per query). We calculated support based on documents retrieved by such a query. The intuition behind this measure is that if there are many documents that mention the two entities together in close proximity, then this evidence of high co-occurrence may indicate some relationship between the two entities, even if there is no explicit mention of the relation asked for. A good example here is query number 3 in test topics, that looked for all "students of `Claire Cardie`". In this case, DBLP pages, and other university pages came out as valid sources to

Particulars ⇓ Run ID ⇒	UIauto	UIqryForm	UIqryForm3
Type of run	Automatic	Manual	Manual
Type of manual input	None	Removed "digressing synonyms" from the relation queries	
Number of candidates returned	As many	As many	Only top 3
Number of relevant documents considered	100	20	10
Re-ranking rules			
• Support	Yes	Yes	Yes
• Relatedness	Yes	No	No
Special processing of Wikipedia pages	No	No	No
External Resource used			
• Distributional similarity	Yes	Yes	Yes
• WordNet	No	Yes	Yes
Handling Named Entities	LBJ-Named Entity tagger and Product recognizer		

Table 1: Parameters of the submitted runs

justify relation between `Claire Cardie` and her students, even though it did not mention the advisor-student relationship on the page.

We combined the two scores using a simple weighted sum, with the relative weight of importance manually set to prefer support over relatedness in the ratio of 2:1.

As the final step, we searched the corpus for homepages of the final, ranked list of candidate entities. This was done using a simple IndriQuery that weighted the occurrence of the candidate name in the title and headings higher than the body (in the ratio of 5:4:1, respectively). We pruned the retrieved set of results to only consider the top 3 documents as homepages, as per the submission guidelines. If one of the top 3 pages returned was a Wikipedia page, then we assigned that document id to the optional "Wikipedia homepage" field and continued up to the top 4 documents instead of 3. A shortcoming of this approach was that a document might come as a top-ranked result for many entities if it is significant for multiple candidates. For example, the DBLP page of one of the co-authors might have multiple occurrences of other candidate entities, and hence might come as a top retrieved document for many entities. Since finding homepages was not the primary focus of the research ideas we wanted to explore, we applied simple rules to check that the resultant pages do not repeat under one query topic, again to follow the submission guidelines.

The process of finding homepages also acted as the final pruning step. If we could not find any document returned for a candidate entity, we dropped that candidate from the final list. This was also mandated by the submission guidelines.

Once the final ranked list of candidates was decided, we collected all documents that supported these candidates from the original list of documents returned for the relation query. This subset was mentioned as the list of supporting documents in the submitted runs.

3. DETAILS OF THE SUBMITTED RUNS

We submitted three runs, viz. `UIauto`, `UIqryForm`, and `UIqryForm3`; and they are summarized in Table 1. None of the runs handled Wikipedia sub-collection in any special way. The details about the runs are explained below.

3.1 UIauto

This was an automatic run. We constructed the relation query with only the words in the narrative. We did not use WordNet, but considered only the closest word from the distributional similarity-based word cluster as the synonym. The top 100 documents were retrieved and the relevant snippets were passed through the named entity recognizers. Then, the candidate entities were re-ranked based on both support and relatedness metrics.

3.2 UIqryForm

This run was manual. We first applied WordNet based synonym expansion to the relation word. However, we found that it added many words that drifted the intent of the query. So, we have to manually prune out the other irrelevant senses from the expansion list. We considered the top 20 documents and identified entities from the snippets as described earlier. The entities were ranked only on the support metric. All the short-listed entities were returned.

3.3 UIqryForm3

This run was also manual. It was basically the same setup as `UIqryForm`, except that only the top 10 documents were considered while finding related entities, and only top 3 entities were returned for each query.

4. EVALUATION

As per the official results released, the `UIauto` run seemed to perform the best among the runs we submitted. The results are summarized in the first row of Table 2.

Since our primary focus was to check if we are able to retrieve the related entities, and not on finding homepages corresponding to the related entities, we also evaluated our performance on recognizing relevant named entities alone. For this, we considered all the name strings identified in the released *qrel* file for this track and computed the retrieval measures over the names of the entities. It must be noted that a name was judged correct if it matched up with something else in the record, even if the record was neither primary nor relevant for the topic [1]. This means that, for the purpose of evaluating names alone, the *qrel* file may include spurious entries. We were not aware of this anomaly

Particulars ⇓ Run ID ⇒	UIauto	UIqryForm	UIqryForm3
nDCG (official)	0.0575	0.0302	0.0189
# Named Entities (NEs) returned	624	510	57
# relevant NEs returned	139	89	28
Micro-averaged precision	0.2228	0.1745	0.4912
Macro-averaged precision	0.3876	0.3671	0.5083
Precision @ 5	0.5400	0.4200	0.2800
Precision @ 10	0.4050	0.2350	0.1400
Precision @ R (R: # retrieved NEs)	0.3876	0.3671	0.5083
Precision @ 5 (adjusted for R)	0.6083	0.5083	0.5083
Precision @ 10 (adjusted for R)	0.5158	0.4018	0.5083

Table 2: Summary of results

till the TREC workshop and we could not find an automated approach to easily overcome this.

Table 2 summarizes our performance on retrieving relevant named entities. In the table, micro-averaged precision is calculated by accumulating counts of relevant and retrieved entities over all query topics, and then finding the precision over the accumulated counts. In contrast, macro-averaged precision is calculated by first computing the precision for each query topic and then taking their average.

As shown in Table 2, we get a macro-averaged precision of over 0.36 for both `UIauto` and `UIqryForm`. In `UIqryForm3`, since we restricted our results to only top 3 named entities for each topic query, the macro-averaged precision was significantly higher at 0.51, but at the cost of lower number of returned entities. This probably suggests that we were more successful in finding the relevant entities, but could not get valid homepages for most of the entities we found. The Precision at 5 and Precision at 10 measures also show that `UIauto` run performed better over the other runs. It is able to achieve an average of 60% precision, computed at 5 results, adjusted for number of entities retrieved.

In our experiments, when we added WordNet based synonyms, we added too many digressing words, leading to poor results. In the submitted runs `UIqryForm` and `UIqryForm3`, we manually pruned the expansion list to only retain valid synonyms. So, the main reason for poor performance of the manual runs probably comes from the fact that these use fewer documents to find related entities. Additionally, these two runs did not use relatedness measure to further improve confidence of the extracted entities.

5. DISCUSSION

Large corpora often test the existing tools and system implementations to the core, for both scalability and robustness. This was the case also with the ClueWeb09 corpus. During the initial stages, we wanted to explore if we could parse the corpus with NLP tools such as the Named Entity tagger and co-reference resolver. This way we could use special indexes over the named entities for better retrieval, we hoped. But processing the large corpus took inordinate time, so we had to revert to applying NLP tools only on the smaller snippets, increasing the query time.

Another important issue was to overcome the errors of existing NE taggers and filters. On analyzing the candidates, we saw that the two main classes of errors that the taggers produced were erroneous classification (identifying the wrong named entity type) and to a lesser extent, identifying partial entities. We tried to overcome these by also including entities marked as `MISC` to overcome the type-naming errors. This was especially important for the product class of queries. We tried to overcome the partial entity recognition problem with edit-distance based clustering of similar candidate names.

6. CONCLUSIONS

To find the related entities from text, we have tried to incorporate NLP resources to analyze deeper semantics of text rather than surface matches. Our experiments validated our assumption that it is possible to find related entities using NLP techniques such as relation identification. Further, the results show that ranking related entities is improved by finding other supporting relations between entities from text. However, the manual improvements to the relation extraction stage does not show improvement; and further experimentation and analysis would be required to better understand the reason. The performance can be further improved with better techniques to filter false-positive entities and improve the ranking based on external corpora, such as Wikipedia. We believe that IR methodologies can help from deeper natural language processing and hope to continue this direction of research of combining NLP tools and techniques for improving focused searches.

7. REFERENCES

[1] K. Balog, A. P. de Vries, P. Serdyukov, P. Thomas, and T. Westerveld. Overview of the TREC 2009 Entity Track. In *Proceedings of the Eighteenth Text REtrieval Conference (TREC 2009)*, 2010.

[2] D. M. Bikel, R. Schwartz, and R. M. Weischedel. An Algorithm that Learns What's in a Name. *Machine Learning Special issue on Natural Language Learning*, 34(1-3):211–231, 1999.

[3] P. F. Brown, V. J. D. Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479, 1992.

[4] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[5] D. Lin. Automatic Retrieval and Clustering of similar words. In *COLING-ACL*, pages 768–774, 1998.

[6] L. Ratinov and D. Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[7] D. Roth, M. Sammons, and V. Vydiswaran. A Framework for Entailed Relation Recognition. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 57–60, Suntec, Singapore, August 2009. Association for Computational Linguistics.

[8] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language-model based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, 2005.